

# A Three-Level Approach to Texture Mapping and Synthesis on 3D Surfaces

KERSTEN SCHUSTER, PHILIP TRETTNER, PATRIC SCHMITZ, and LEIF KOBBELT,  
Visual Computing Institute, RWTH Aachen University

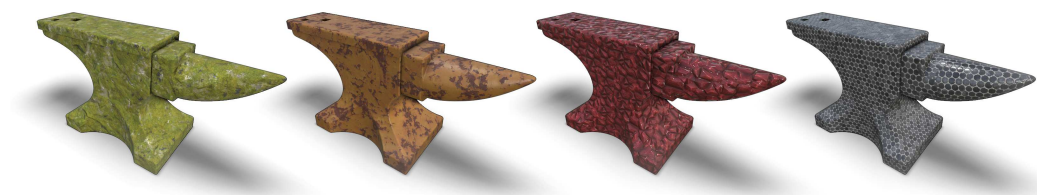


Fig. 1. Our method enables automatic texturing of triangular meshes with low distortion and minimal visual artifacts. The supported texture types range from stochastic over irregular to quasi-regular materials. Model by [Thingiverse 2012] and textures from [Textures.com 2020].

We present a method for example-based texturing of triangular 3D meshes. Our algorithm maps a small 2D texture sample onto objects of arbitrary size in a seamless fashion, with no visible repetitions and low overall distortion. It requires minimal user interaction and can be applied to complex, multi-layered input materials that are not required to be tileable. Our framework integrates a patch-based approach with per-pixel compositing. To minimize visual artifacts, we run a three-level optimization that starts with a rigid alignment of texture patches (macro scale), then continues with non-rigid adjustments (meso scale) and finally performs pixel-level texture blending (micro scale). We demonstrate that the relevance of the three levels depends on the texture content and type (stochastic, structured, or anisotropic textures).

CCS Concepts: • **Computing methodologies** → **Texturing; Mesh geometry models; Interest point and salient region detections; Matching; Image processing; Rendering.**

Additional Key Words and Phrases: surface texture synthesis, texture mapping, material blending

## ACM Reference Format:

Kersten Schuster, Philip Trettner, Patric Schmitz, and Leif Kobbelt. 2020. A Three-Level Approach to Texture Mapping and Synthesis on 3D Surfaces. *Proc. ACM Comput. Graph. Interact. Tech.* 3, 1, Article 1 (May 2020), 19 pages. <https://doi.org/10.1145/3384542>

## 1 INTRODUCTION

High-quality texturing of graphical assets is essential to achieve visual realism in computer graphics. Sophisticated lighting and physically-based shading are the de-facto industry standard today. Realistic texturing of objects, however, still requires a considerable amount of manual work. While many models exhibit large areas of homogeneous material, no automatic method exists to apply an input exemplar to a 3D object without compromising visual quality.

---

Authors' address: Kersten Schuster, [schuster@cs.rwth-aachen.de](mailto:schuster@cs.rwth-aachen.de); Philip Trettner, [trettner@cs.rwth-aachen.de](mailto:trettner@cs.rwth-aachen.de); Patric Schmitz, [patric.schmitz@cs.rwth-aachen.de](mailto:patric.schmitz@cs.rwth-aachen.de); Leif Kobbelt, [kobbelt@cs.rwth-aachen.de](mailto:kobbelt@cs.rwth-aachen.de), Visual Computing Institute, RWTH Aachen University, Ahornstraße 55, Aachen, 52074, Germany.

---

© 2020 Copyright held by the owner/author(s). Publication rights licensed to ACM.

This is the author's version of the work. It is posted here for your personal use. Not for redistribution. The definitive Version of Record was published in *Proceedings of the ACM on Computer Graphics and Interactive Techniques*, <https://doi.org/10.1145/3384542>.

Several problems can emerge during texture mapping. Length and angle distortion occur when mapping from the 2D texture domain to a 3D surface with non-zero Gaussian curvature. If the map contains discontinuities, seams between portions of the texture become apparent. Furthermore, noticeable repetitions drastically reduce the plausibility of the result. Lastly, ghosting artifacts can become apparent when different regions are blended without prior content alignment.

Existing methods for automatic texturing with an input material suffer from at least one of the aforementioned problems. Per-pixel synthesis produces seamless results, but struggles in preserving large-scale structure from the input texture. Current patch-based methods suffer from blending artifacts or visual discontinuities, while input texture repetitions can still be apparent. Solid synthesis has similar limitations as per-pixel synthesis regarding faithful reproduction of the input structure. Global parametrization approaches can produce seamless texture maps, but require a tileable input texture and produce visible repetitions. Recently, triplanar mapping using histogram-preserving blending was proposed and produces convincing results for stochastic input material [Heitz and Neyret 2018]. For structured or semi-structured textures, however, blending artifacts occur due to content misalignment.

Our automatic texturing method unifies a patch-based approach with per-pixel compositing. Target objects are decomposed into circular patches by a geodesic region growing approach that follows geometric features. These patches are enlarged to form overlap regions before mapping each into the 2D plane with low distortion. Each patch interior samples an unmodified source region from the input texture to reproduce the exemplar as faithfully as possible.

Overlap regions are then aligned directionally, and warped with respect to texture content to facilitate blending without artifacts. Patch alignment happens in two stages. In a global pre-alignment step, orientations of neighboring patches are adapted. Each patch is then non-rigidly transformed with respect to image features in the local patch neighborhood by robustly fitting a transformation that minimizes distances between corresponding interest points.

Finally, patches in overlapping regions are blended in a content-sensitive way to prevent ghosting and contrast reduction. The method minimizes all mentioned types of texturing defects by addressing their causes at all scales: globally, locally between patches, and per pixel. In summary, we focus on smoothly aligning neighboring patches, geometrically and content-wise, to facilitate artifact-free blending operations.

Our main contributions are

- a multi-scale by-example texturing method for 3D surfaces that combines parametrization, patch-based synthesis, and low-artifact blending techniques,
- a content-aware soft registration to align overlapping patches in texture space,
- a multi-pass method to render the result in real-time without requiring texture atlases.

## 2 RELATED WORK

Much research has been conducted in the field of exemplar-based texture synthesis, ranging from pixel-based synthesis [Efros and Leung 1999; Lefebvre and Hoppe 2005; Wei and Levoy 2000] to parametric models [Heeger and Bergen 1995; Portilla and Simoncelli 2000] and, more recently, data-driven approaches [Gatys et al. 2015; Jetchev et al. 2016; Ulyanov et al. 2016]. Due to the vast amount of texture mapping and synthesis techniques, we restrict ourselves to methods that address texturing of 3D meshes or show similarities to our approach. For a more extensive overview, we recommend surveys on general exemplar-based synthesis [Wei et al. 2009] and solid texture synthesis [Pietroni et al. 2010].

## 2.1 Pixel-Based Synthesis

Per-pixel synthesis techniques incrementally determine output texture values by choosing pixels with similar neighborhood from the exemplar. Several approaches texture mesh surfaces by synthesizing per-vertex colors directly on the input mesh [Han et al. 2006; Tong et al. 2002; Turk 2001; Wei and Levoy 2001; Zhang et al. 2003]. Other methods synthesize in the texture domain and compensate distortion of the mapping by warping with its Jacobian and aligning with a tangential field [Lefebvre and Hoppe 2006]. Gorla et al. combine a disjoint triangle patch generation, similarly to ours, with a costly pixel-based synthesis on flattened patches [Gorla et al. 2001]. Instead of aligning neighboring patch orientations, the authors pre-compute rotated copies of the input texture to account for mesh curvature.

Although pixel-based algorithms yield convincing results and can achieve interactive frame rates for small textures [Han et al. 2006; Lefebvre and Hoppe 2006], memory requirements and performance do not scale well with increasing input sizes. Furthermore, unwanted color variations can occur and absence of prominent features in the input texture can lead to low-variance regions in the synthesized results.

## 2.2 Patch-Based Synthesis

In contrast to pixel-based approaches, patch-based methods arrange larger clusters of pixels. Borders of neighboring patches are then concealed, e.g. by optimizing the location of the boundary seam [Efros and Freeman 2001; Kwatra et al. 2003], applying Poisson-blending [Pérez et al. 2003] or combining both approaches [Jia et al. 2006]. One advantage of patch- over pixel-based methods is guaranteed coherence inside the patch and thus preserved input structure.

Lasram and Lefebvre propose parallelized 2D synthesis using circular patches with polar-space circular cuts over which colors are smoothly propagated to minimize feature deviations [Lasram and Lefebvre 2012]. In [Praun et al. 2000], a unique texture patch is repeatedly pasted on the surface and warped according to a direction field, ignoring noticeable repetitions on the target mesh. Schmidt et al. build upon that work to provide a basic texturing and enable an interactive user-guided decal placement, focusing on manual deformation rather than automatic alignment [Schmidt et al. 2006].

## 2.3 Solid Synthesis

Solid texture synthesis is the 3D equivalent of 2D pixel-based synthesis. Colors are synthesized on a voxel grid by simultaneous 2D operations along the three coordinate planes, while taking the voxel's 3D neighborhood into account [Kopf et al. 2007]. The method was later improved by Dong et al. to pre-compute consistent 2D-triplets from the three exemplars, which facilitates near-interactive frame rates in a GPU implementation [Dong et al. 2008]. To synthesize example-based 3D-printable structures, Dumas et al. project rotated input exemplars onto a voxelized surface mesh before hierarchically optimizing structural coherence in a 3D neighborhood [Dumas et al. 2015]. Solid synthesis is convenient for texturing, as it enables using 3D vertex positions as texture coordinates but suffers from the same limitations as pixel-based synthesis.

## 2.4 Global Parametrization

For disk topology meshes, global parametrization algorithms like *Least-Squares Conformal Maps* (LSCM) [Lévy et al. 2002] or *Scalable Locally Injective Mappings* (SLIM) [Rabinovich et al. 2017] yield contiguous texture coordinates, which enables direct mapping of an input exemplar. However, texture seams occur if the parametrization requires cuts, which is the case for every closed mesh.

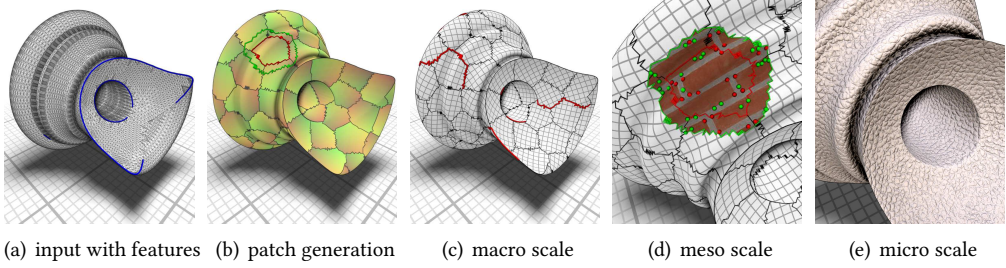


Fig. 2. Method overview: (a) Triangular input mesh with feature edges (blue). (b) Patches with overlap regions and parametrizations (red: core region boundary, green: overlap region boundary). (c) Macro scale: global alignment (red: “fault lines”). (d) Meso scale: content-aware per-patch warping (red, green: interest points). (e) Micro scale: final blending. Model based on [Thingiverse 2011b] and textures from [Textures.com 2020].

In the case of quadrangular remeshing, translational offsets across seams are required to be integer [Campen et al. 2015], so that tileable textures can be mapped seamlessly and in a globally consistent fashion. While this works well for textures consisting only of regular patterns, any irregularity will result in disruptive repetitions. Our method draws on quadrangulation algorithms [Bommes et al. 2013] to determine a globally smooth patch alignment that is invariant to discrete rotations about a symmetry angle.

## 2.5 Procedural Textures

Pioneering work on procedural 3D textures was presented by Perlin [Perlin 1985] to assign color values directly to 3D positions. Unfortunately, procedural algorithms have to be designed specifically for particular classes of textures and do not provide the possibility for local adaptations of the result. Creating realistic 3D equivalents remains a challenge for many types of input exemplars, although plausible results were obtained for stochastic [Galerne et al. 2012; Lagae et al. 2010] and near-regular textures [Gilet et al. 2014].

## 3 METHOD

Our method follows a texture splatting metaphor. The target mesh is divided into overlapping patches, where each patch maps to a different portion of the input texture. Many textures have prominent directions that need to be aligned globally and come in many forms, from stochastic over semi-structured to fully regular. We thus first optimize, on the *macro scale*, the directional alignment between all neighboring patches. Structures and patterns in the overlap regions can, however, still match poorly. This would lead to poor quality of the pixel-level synthesis on the *micro scale*. To create good starting conditions for the micro step, we introduce a content-aware optimization on a *meso scale*. It performs per-patch non-rigid warping to maximize the content similarity between neighboring patches.

By copying continuous chunks of the input texture, randomizing texture coordinate offsets, and creating optimal conditions for pixel-level compositing, we maximize overall texture coherence and quality. Structure in the input exemplar is preserved at patch size and below, while texture repetition and the forming of structures above patch size is prevented.

Figure 2 presents an overview of our method and Figure 3 shows how the user parameters of our method can be interpreted in the texture space. The steps are presented in detail in the subsequent sections of this chapter.



### 3.1 Patch Generation

Decomposition of the surface into overlapping, evenly distributed and near-isotropic regions is performed in several steps (cf. Figure 4).

**3.1.1 Preprocessing.** Aside from the texture sample, the user provides a triangular mesh and optionally a set of feature edges as input. If no feature edges are provided, they are selected automatically if the corresponding dihedral angle is above a given threshold. Patches are assigned per-face and large triangles might reduce patch uniformity. Thus, we split all edges that are longer than half the overlap radius.

Patches are prevented from growing over feature edges, which serves two purposes. First, they prevent patches from growing over corners, as those regions cannot be unwrapped with low distortion. Second, they provide candidates for the “fault lines” that the macro step can introduce.

**3.1.2 Layout Generation.** Surface partitioning starts with a low-discrepancy sampling of patch centers, followed by an approximate centroidal Voronoi tessellation (CVT) [Du et al. 1999]. The resulting patches are distributed with near-uniform distance and their size is bounded by the desired maximum patch radius  $r_p$ . All distance computations are performed approximately by a fast marching technique [Novotni et al. 2002].

Patch centers are initialized as follows:

- (1) Pick a random unassigned triangle as seed.
- (2) Grow a surface patch up to a maximum distance  $r_p$ .
- (3) If unassigned triangles exist, go back to (1).

During step (2), already assigned triangles can be assigned to a new patch if the distance to the new seed is shorter, effectively computing a discrete Voronoi diagram.

We then perform a few Lloyd relaxation steps to approximate a CVT, noticeably improving patch uniformity. Concretely, for each patch we select the patch triangle with largest distance to the patch boundary as new seed and then re-grow all patches.

For non-tileable input textures, the patch radius  $r_p$  is bounded by the desired texture size in world space and is further constrained by the goal to reduce prominent repetitions. On the other hand, patches need to be sufficiently large to capture structural elements within the input texture (e.g. cells, bricks, patterns).

It is crucial to define overlap regions to enable content-aware alignment of patches and perform the final per-pixel compositing. The region has to cover the typical size of texture features but should otherwise be kept as small as possible. For every patch  $p$ , we re-grow the initial core region with an appropriately enlarged patch radius.

**3.1.3 Texture Coordinate Computation.** Each patch needs to be mapped into the 2D plane, while distortion that arises from non-zero Gaussian curvature should be distributed as evenly as possible. Using the *Least-Squares Conformal Maps* algorithm (LSCM) [Lévy et al. 2002], most patches can be mapped without visible distortion. Solutions can be found very efficiently by directly solving a system of linear equations.

However, patches with high-curvature regions often result in noticeable length distortion. In those cases, minimizing an isometric deformation energy that encourages length preservation is preferable. A prominent example is the symmetric Dirichlet energy. While it is more expensive to

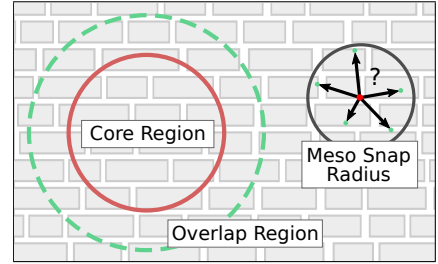


Fig. 3. User parameters: size of core and overlap regions, interest point snap radius.

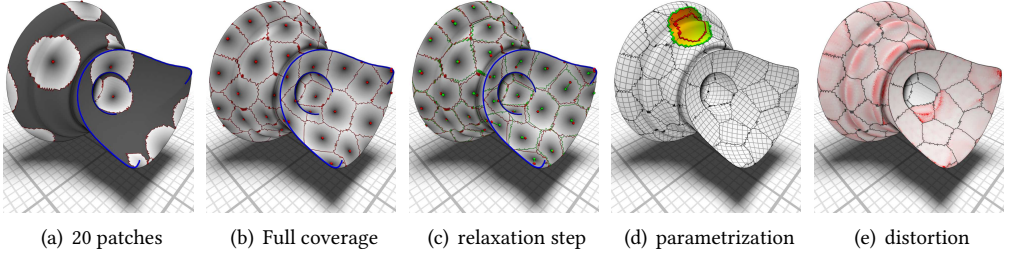


Fig. 4. Patch layout generation: (a, b) geodesic growing from Poisson-sampled seeds (blue: feature edges). (c) Lloyd relaxation steps for uniformity. (d) Overlap region growing (green) and per-patch parametrization. (e) Resulting length distortion (white: 0%, red: 5%). Model based on [Thingiverse 2011b].

minimize than the quadratic LSCM energy, specialized numerical methods such as *Scalable Locally Injective Mappings* (SLIM) [Rabinovich et al. 2017] exist.

In order to exploit the performance of LSCM and the stretch-minimizing properties of SLIM, we combine both methods to compute our mapping:

- (1) Apply LSCM to all core regions.
- (2) For patches that still exhibit considerable distortion we apply SLIM on the LSCM result.

This procedure is repeated for the overlap regions while keeping the core region vertices fixed. As a result, distortion tends to concentrate in overlap regions where it is concealed by the final blending operation.

Finally, we compute blending weights for all vertices in the overlap region. Every vertex that touches at least one core region face is assigned a weight of 1 and every vertex that belongs to the patch boundary has weight 0. For vertices in-between we compute a harmonic scalar field (cf. Figure 5(a)) such that weights vary smoothly between 1 and 0 with increasing distance to the core region (cf. Figure 5(b)).

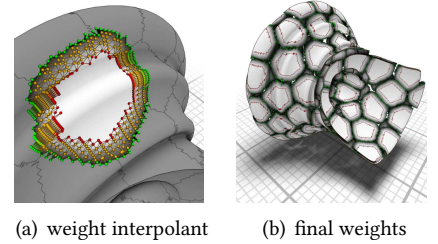


Fig. 5. Blending weight interpolation (green: overlap border, red: core border). Model based on [Thingiverse 2011b].

### 3.2 Macro Scale

In the case of textures with considerable directional structure, an alignment of patch orientations is essential. We optimize neighboring per-patch rotations and successively exclude problematic patches. To synthesize textures with prominent structure, we find it more natural to relax the directional alignment at a few “fault lines” (e.g. at geometric features) than to distribute the alignment error over the whole surface. In other words, we prefer many well-aligned patches with a few bad outliers to a least squares solution.

The problem is formulated globally and serves as a starting point for the *meso optimization* step. To this end, we adapt the mixed-integer programming formulation introduced by [Bommes et al. 2009]. Angular differences are minimized across all core region boundary edges, up to integer multiples of the optional symmetry angle  $\theta_s$ . We optimize for per-patch orientations  $\theta_i, \theta_j$  that share a set of boundary edges  $E_{ij}^b$ . Angles are computed relative to a local coordinate frame that is determined by flattening the adjacent triangles along their common edge. Since patches share multiple edges, we compute an average angle difference  $\kappa_{ij}$  between the local coordinate frames.

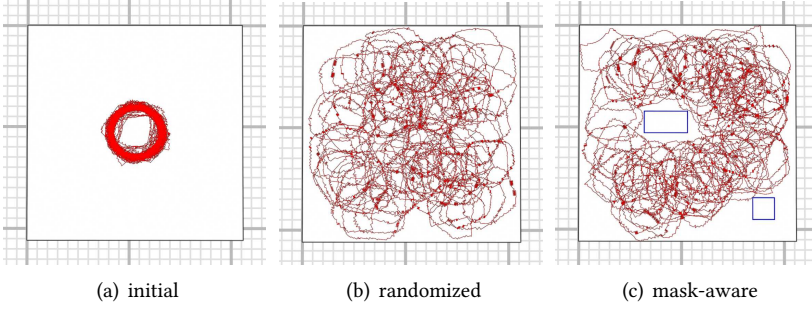


Fig. 6. Texture space distribution of patches: (a) Centered initial parametrizations. (b) Random distribution to reduce repetitions. (c) Mask-aware rejection sampling to avoid unwanted regions.

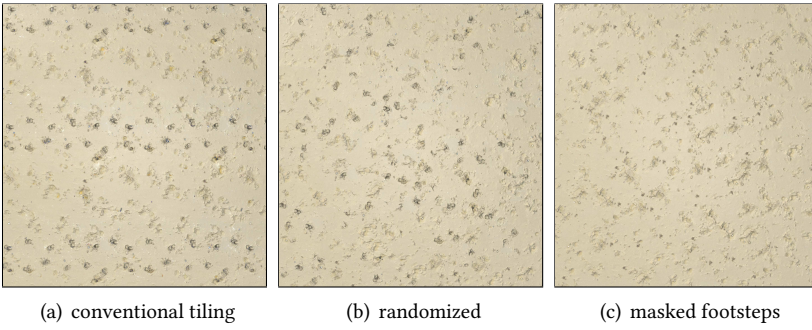


Fig. 7. Comparison between texture tiling and our patch distribution. (a) Tiling exhibits repetitions. (b) Random distribution de-correlates patches. (c) Footsteps in the sand are masked. Texture from [Textures.com 2020].

For the final optimization energy, we weight the angular differences between each pair of patches by the length  $l_{ij}$  of their shared boundary.

$$E_{\text{macro}} = \sum_{ij} l_{ij} (\theta_i - \theta_j + \kappa_{ij} + p_{ij} \theta_s)^2 \quad (1)$$

where  $p_{ij}$  is the integer-valued period jump between the patches. The energy is then minimized using a greedy-rounding strategy and constraint elimination [Bommes et al. 2010] in cases where the user fixes the orientation of some patches.

Boundary edges that were marked as “fault lines” are not considered in the above functional. If the user specified a maximum angular deviation instead, all edges that exceed the threshold are removed from the respective set  $E_{ij}^b$  and the process is iterated until a solution is found.

### 3.3 Meso Scale

After patch generation and macro optimization, the surface is covered by roughly aligned, overlapping patches. In the final micro step, the overlap regions will be blended locally to create a seamless texture. If the texture contains prominent interest points or structures that are not aligned, the blending often performs poorly, resulting for example in ghosting artifacts. While the parametrization step minimizes texture distortion, the meso step may reintroduce a small amount of distortion by bending and warping patch texture coordinates. This improves content similarity between

overlap regions and provides a better starting point for the final blending operations. Note that before the meso alignment takes place, it is possible to rotate the texture coordinates of all patches by a constant angle without introducing any further distortion, enabling the user to change the global orientation.

As 2D texture content should be matched, parallels to image registration are apparent. However, we aim for aligning similar structures rather than finding perfect matches which would lead to visible repetitions in the case of tiled textures and an unsolvable problem if textures are not tileable. To avoid confusion with geometric feature edges, we call the texture structures that we want to align *interest points*. Such interest points can either be provided by the user or derived automatically, for example by computing Harris corners [Harris et al. 1988] (cf. Figure 8). While this works well for near-regular and regular structures, stochastic and irregular ones usually do not benefit from it.

As depicted in Figure 6(a), patches overlap strongly in the texture domain after initial parametrization. Prior to the meso optimization step, we randomize patch location and rotation to reduce overlap and break up repetitions and structures bigger than a single patch. Rotation must be compatible with the macro optimization, i.e. if a directional alignment was performed, patch rotation can only be adjusted by multiples of the symmetry angle. Translation is less constrained. If the texture is not tileable, patches are not allowed to leave the  $[0, 1]^2$  texture space region. Furthermore, some textures might contain impurities or unwanted features that a designer might want to “mask out”. We found that a simple rejection sampling approach suffices: randomize patch transformation (potentially constrained by the macro optimization) until no masked region constraints are violated. The results are shown in Figure 6 and 7.

The meso optimization works iteratively. For each patch  $p$  we keep all neighboring patches  $p_i$  fixed and try to find a small transformation of texture coordinates that improves the structural alignment in the overlap regions. Patches are processed in a random order and theoretically until convergence, though we stop after a few iterations in practice when changes become too small.

Formally, we have a discrete set  $I \subset [0, 1]^2$  of interest points in the texture domain of patch  $p$ , stored in a  $k$ -d tree for faster lookup. In the overlap region, each interest point  $q_a$  of each neighboring patch  $p_i$  is mapped to the texture domain of  $p$  and all interest points  $q_b$  inside a radius  $r_i$  are collected (see Figure 2(d), red dots are interest points of  $p$ , green dots of the  $p_i$ ). Each pair  $(q_a, q_b)$  represents a potential correspondence. Our goal is to find a non-rigid transformation  $T$  that maximizes the number of correspondences for which  $\|T(q_a) - q_b\| < \epsilon$ .

This is similar to the ICP algorithm [Besl and McKay 1992] or classical RANSAC [Fischler and Bolles 1981] with some important differences and difficulties:

- we determine  $T$  as a cubic transformation for non-linear warping
- there is no “ground truth”, a perfect match is not achievable
- we assume that the number of correspondences belonging to the “best” transformation is low

The cubic transformation has 20 degrees of freedom and thus requires at least 10 correspondences to be fully defined. In the RANSAC framework, the probability to pick 10 correspondences belonging to the “correct” transformation is so low that a prohibitive number of iterations is required.

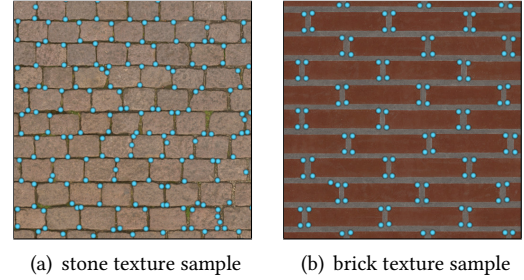


Fig. 8. Interest point detection (blue: Harris corners). (a) Near-regular structure. (b) Regular structure. Textures from [Textures.com 2020].

Instead we opt for a different approach:

- (1) choose three potential correspondences
- (2) solve a regularized linear system for an initial  $T'$
- (3) collect all inliers belonging to transformation  $T'$
- (4) compute  $T$  as the least-square fit of all inliers

A cubic transformation  $T(u, v)$  consists of a cubic polynomial

$$T(u, v) = c_0 + c_1u + c_2v + c_3u^2 + c_4uv + c_5v^2 + c_6u^3 + c_7u^2v + c_8uv^2 + c_9v^3 \quad (2)$$

with 2D coefficients  $c_0$  to  $c_9$ . Let  $x \in \mathbb{R}^{20}$  contain all coefficients and  $\|Ax - b\|$  be the linear regression system for fitting correspondence pairs  $(q_a, q_b)$ . The identity transformation  $n$  corresponds to  $c_i = (0, 0)$ , except  $c_1 = (1, 0)$  and  $c_2 = (0, 1)$ . The regularized system that we optimize in step (2) is

$$\|Ax - b\| + \|W \cdot (x - n)\| \rightarrow \min \quad (3)$$

where  $W$  is a diagonal matrix with weights that penalize quadratic and cubic coefficients more than linear and constant ones, to bias the system towards stiff transformations. The exact penalty weights do not matter much and we found 0 for translation, 1 for scale, 0.01 for quadratic, and 0.1 for cubic to be a good compromise.

### 3.4 Micro Scale

So far we have optimized the alignment of patches in a global and local fashion. Due to mesh curvature and potentially challenging texture structure, optimized compositing of overlap regions is required. This task can be approached in 2D since all patches have been flattened. Three methods are typically used to address seams in planar patch-based texture synthesis:

- blending (e.g. [Pérez et al. 2003])
- seam optimization (e.g. [Kwatra et al. 2003])
- pixel-based re-synthesis (e.g. [Nealen and Alexa 2003])

All those approaches have their own advantages and disadvantages. While seam optimization and pixel-based re-synthesis are comparatively costly, ordinary linear blending reduces the contrast of the result and can produce ghosting artifacts. This becomes even worse if the number of blended values increases [Heitz and Neyret 2018]. With all our pre-registration and alignment optimizations, we created favorable conditions for blending-based approaches, which we prefer in our method (though the other approaches can be used as well). In particular, we show results using the following two approaches:

**3.4.1 Histogram-Preserving Blending.** Recently, Heitz and Neyret presented a histogram-preserving blending method that overcomes the downsides of linear blending by mixing exemplars in a *gaussianized* color space before re-adjusting the reduced variance and converting back to RGB space [Heitz and Neyret 2018]. Deliot and Heitz further improved this method, especially for reduced pre-processing time [Deliot and Heitz 2019], and Burley proposed additional adaptations to reduce clipping, coloration and ghosting artifacts [Burley 2019].

**3.4.2 Min- and Max-Blending.** Texture sets for physically based rendering (PBR) typically consist of at least albedo, roughness, normal, and height maps. The purpose of blending is now to transition between two materials. Given height information, a natural way to transition is to define a height function that is modulated by blending weight and then choose the material with the highest (or lowest) height, which we call *max-blending* and *min-blending* respectively. This is particularly useful for textures where salient structures protrude.



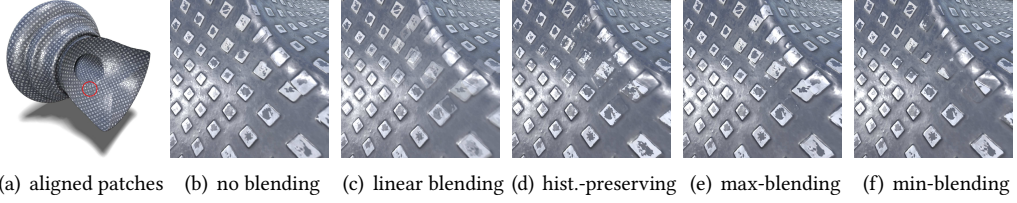


Fig. 9. Comparison of blending modes: (a) Macro view of an object after alignment. (b-f) Detail view of the region marked in red. Model based on [Thingiverse 2011b] and texture from [Textures.com 2020].

Formally, we define the effective height  $h_e$  as  $w \cdot h$  (*max-blending*) or  $w \cdot (1 - h)$  (*min-blending*), where  $w$  is linearly interpolated per-vertex weight and  $h$  the sampled height map. Material parameters are then taken from the sample with highest effective height. To prevent hard edges, a small “grace”  $\Delta h$  is defined and all samples with  $h_e \in [h_{e,\max} - \Delta h, h_{e,\max}]$  are composited with histogram-preserving linear blending.

A comparison of the blending strategies is shown in Figure 9. While omitting blending completely creates visual seams (b), linear blending introduces salient ghosting artifacts (c). These can be reduced with histogram-preserving blending (d). If a height map is available, patches with higher (e) or lower (f) structures can be preferred.

### 3.5 Storage and Rendering

There are two main ways to use the output of our method: *direct rendering* and *atlas baking*.

In *direct rendering*, we render each patch (including overlap) using its texture coordinates and perform the blending in screen space using a multi-pass method:

- (1) Perform a *depth pre-pass* using the original geometry.
- (2) Render all patches using maximum blending to write out the effective height (cf. Section 3.4.2).
- (3) Render all patches using additive blending and discard fragments with  $h_e < h_{e,\max} - \Delta h$ , compute new weight  $w = (h_e - \max(0, h_{e,\max} - \Delta h)) / \Delta h$ , and *accumulate* a mini g-buffer (albedo, world space normal, etc.) weighted by  $w$ , as well as  $w$  itself.
- (4) *Resolve* the g-buffer by dividing the accumulated values by the accumulated weight.

Linear blending can be recovered by skipping step 2, setting  $h_e$  to the interpolated patch weights, and  $\Delta h$  to 1. When using histogram-preserving blending, we also write out  $w^2$  as it is needed for the variance preservation and apply the de-gaussianization via look-up texture.

The second approach is a more traditional *atlas baking* which is especially simple for our method (cf. Figure 10). The patch core regions form a partitioning of the surface. Each region already has texture coordinates via parametrization, is roughly spherical and of similar size by construction. We arrange the patches in a regular grid with power-of-two cell size. While a more sophisticated packing could reduce the required space on the highest resolution, textures are usually used with mipmapping and a tighter packing would result in cross-patch blurring on higher mipmap levels. Alternatively, baked patches can be stored in individual layers of a 2D texture array where each patch texture has its own mipmaps.

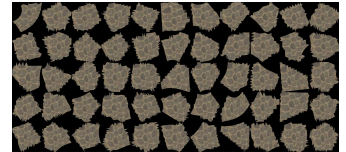


Fig. 10. Example texture atlas. Texture from [Textures.com 2020].

When baking the texture atlas we use the same blending as in the *direct rendering*. Note that we have to apply normal mapping to get world space normals, blend them, and convert them back into

a tangent space defined by the texture atlas coordinates because directly blending normal maps from different tangent spaces is not correct.

*Direct rendering* creates more geometric data because overlap regions have to be duplicated for each patch. It is slower to render because blending has to be performed via a custom multi-pass method in each frame. However it requires no additional preprocessing and no additional texture space, which makes it well suited for interactive visualization and for high-resolution textures. In contrast, a *texture atlas* leverages the traditional texture pipeline with no additional runtime overhead which is useful for low-power devices or for exporting a textured mesh. On the other hand, the required texture size can be well above usual memory limits.

## 4 RESULTS AND DISCUSSION

Results of our method are illustrated in Figure 1 for stochastic, near-stochastic, irregular, and near-regular materials. A larger selection of texturing results can be found in Figure 11 and in the supplemental material. In the following, we motivate each of our optimization steps and discuss it with regard to existing methods. Furthermore, we present exemplary timings and discuss the limitations of our approach.

### 4.1 Discussion

The importance of a global directional alignment, our macro step, is demonstrated in Figure 12 on a near-regular texture with a dominant direction.

In addition, many textures have local features that create ghosting artifacts in the overlap regions if they are not aligned. This affects mostly near-regular and irregular textures, often with cellular structures. As shown in Figure 13, this can often be resolved by our non-rigid registration in the meso step.

While the macro and meso steps improve the alignment for structured textures, the micro step is always important. Even with a good alignment, the irregular and stochastic parts of a texture differ between overlapping patches. Thus, omitting the blending operation results in visible cuts, which can be observed in Figure 14. For stochastic textures, histogram-preserving blending works fine (cf. Figure 11 A, B). It could also be performed by the *triplanar mapping* method proposed by Heitz and Neyret [Heitz and Neyret 2018], provided that the input texture is tileable. Their method is very flexible, as it does not require any preprocessing of the input mesh. However, it produces misalignment artifacts for non-stochastic inputs as well as length distortions, because blending is performed in all surface regions that are not aligned with any of the three blending planes. Figure 15 shows a simple mesh with an irregular texture for comparison. Even though irregular inputs cannot be aligned using the macro and meso optimizations, our method yields more consistent results, since blending occurs only in overlap regions.

This illustrates a general advantage of patch-based over per-pixel methods: patch content, apart from overlap regions, is always coherent and does not require further local adjustment. Pixel-based synthesis methods (2D and 3D) often struggle with low-contrast regions and the absence of prominent features in the exemplar. Furthermore, the performance depends on the size of the input

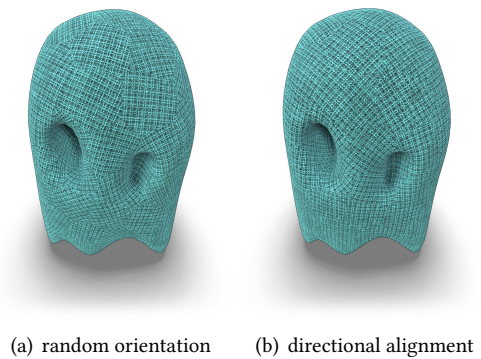


Fig. 12. Macro optimization result. Model from [Thingiverse 2011a] and texture from [Textures.com 2020].



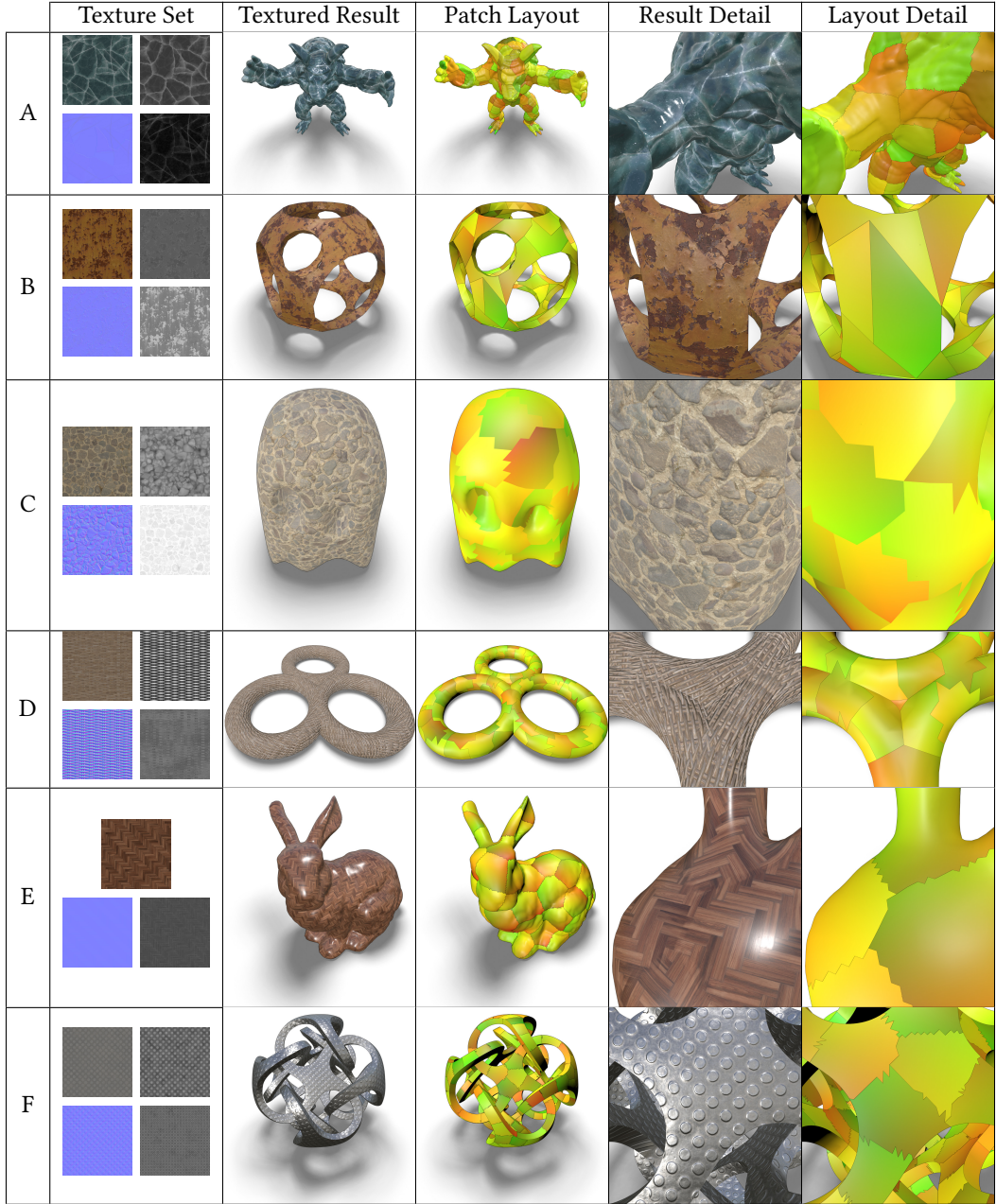


Fig. 11. Results from our synthesis algorithm. The texture sets consist of albedo, roughness, normal, and optional height map if it is used to guide the blending. Texture coordinates are indicated as red and green. Table 1 contains corresponding statistics and timing measurements. High resolution results and videos can be found in the supplemental material. Armadillo and bunny models from [Stanford 2020], ghost model from [Thingiverse 2011a], entangled model from [Thingiverse 2015], textures from [Textures.com 2020].

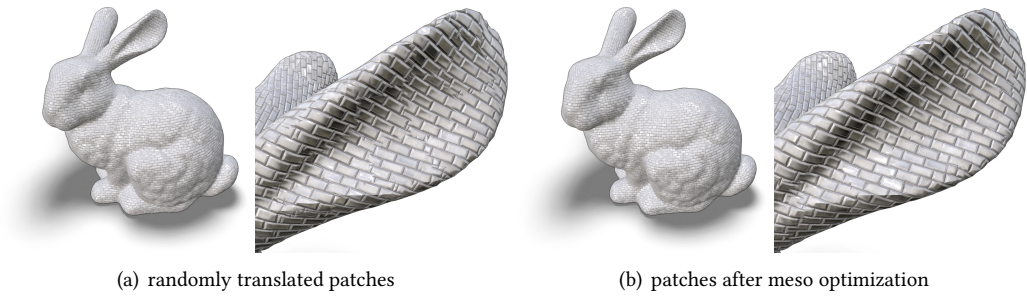


Fig. 13. Meso optimization results: content-aware pairwise alignment reduces visible blending artifacts. Model from [Stanford 2020] and texture from [Textures.com 2020].

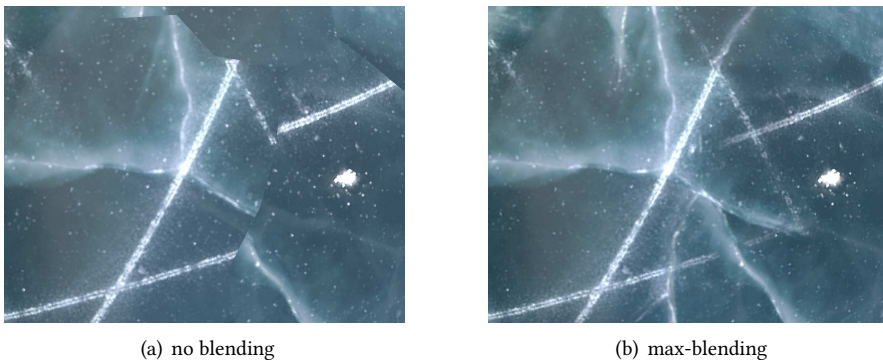


Fig. 14. Micro optimization results: without pixel-level blending, patch borders are visible in the texturing. Texture from [Textures.com 2020].

exemplars and a texture atlas is necessary to store the synthesized result. Our algorithm, however, is independent from the actual texture size, because it only computes per-vertex blending weights and texture coordinates to access the input texture directly.

Although pixel-based methods exhibit the aforementioned limitations, we believe that they could be incorporated into the compositing of overlap areas in our micro optimization. In that case, the use of an atlas would be inevitable. To reduce the memory requirements of the texture atlas, only the overlap regions have to be blended and baked into a texture before re-synthesizing them (using e.g. [Lefebvre and Hoppe 2006]). The core regions would still be textured by directly sampling from the input texture.

A noticeable limitation of our method is fully regular input material, like tilings or mosaics. While the macro optimization adjusts pairwise patch orientations and the meso step aligns neighboring patches, the local warping does not adapt to the surface curvature in a global fashion. In contrast, global parametrization methods (e.g. [Campen et al. 2015]) distribute curvature-induced distortion over the whole mesh, which is preferable if repetitions are explicitly desired and the input texture is tileable.

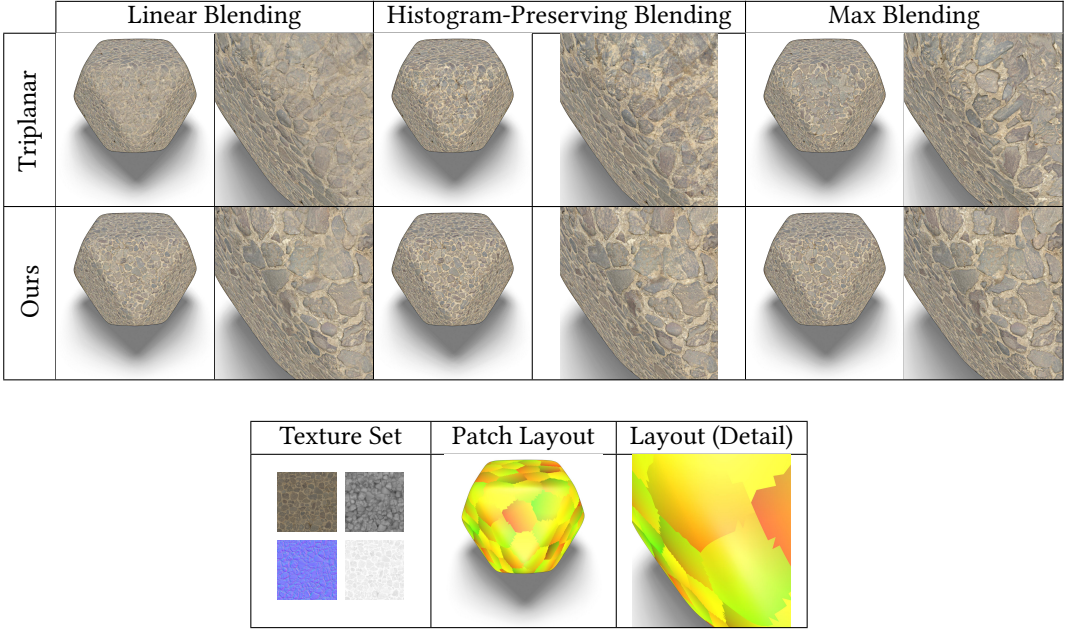


Fig. 15. Top: comparison of triplanar mapping with our method for an irregular stone texture. Bottom: texture set (albedo, height, normal and roughness) and patch layout computed by our method. Textures from [Textures.com 2020].








## 4.2 Performance

Our experiments were run on a 3.20 GHz Intel i7-8700 and used an NVidia GTX 1080 GPU for rendering. The code is written in C++ and was compiled with clang 7 using `-O2`. We use `libigl` [Jacobson et al. 2018] for SLIM, while our LSCM implementation and the weight computations use the `Eigen::SimplicialLDLT` solver from [Guennebaud et al. 2010]. Renderings were created in  $2560 \times 1440$  resolution using the *direct rendering* approach from Section 3.5. The largest texture was of size  $8192 \times 8192$  and was used for the parquet bunny in Figure 11 E.

While there are many opportunities for further optimization, the results in Table 1 provide a rough overview of our method’s performance. Note that the synthesis timings are independent of the texture resolution and the implications on rendering performance are low due to mipmapping.

**4.2.1 Synthesis Time.** Patch generation performance is mainly influenced by the number of input triangles, while most of the time is spent for the Lloyd relaxation. For the given examples, we performed patch growing with 10 relaxation steps. If there are more synthesis faces than input faces, large edges are split to ensure a more homogeneous patch layout. However, even a poor triangulation, which can result in a non-uniform patch layout, often yields convincing results (cf. the CSG ball in Figure 11 B). Parametrization and weight computation performance is influenced by the patch size, i.e. a few patches with many triangles are more costly than many patches with a low triangle count. The macro optimization cost scales with the number of patches and becomes more expensive for smaller angle deviation tolerances specified by the user. In the meso step, alignment costs are highest for textures with a vast number of interest points, as well as a large search radius in which nearby interest points are gathered. For the renderings in the last three columns of Table 1, five meso iterations were performed.

Table 1. Timing measurements for the synthesis and rendering algorithms for various meshes. *Render Faces* consist of the disjoint synthesis faces and the overlap faces. To facilitate blending, overlap faces are rendered once for every patch they are covered by. Armadillo and bunny models from [Stanford 2020], ghost model from [Thingiverse 2011a], entangled model from [Thingiverse 2015], and textures from [Textures.com 2020].

	Fig. 11 A	Fig. 11 B	Fig. 11 C	Fig. 11 D	Fig. 11 E	Fig. 11 F	Fig. 13(b)
							
<b>Statistics</b>							
Input Faces	345 944	1128	3392	3306	69 666	137 302	69 666
Synthesis Faces	345 944	1128	3524	3306	69 666	137 302	70 780
Render Faces	776 602	6078	12 760	8966	150 933	294 593	134 380
Genus	0	13	0	3	0	11	0
Patches	201	200	75	140	72	562	652
<b>Synthesis</b>							
Patch Growing	5727 ms	19 ms	20 ms	21 ms	533 ms	925 ms	514 ms
Parametrization	16 228 ms	16 ms	85 ms	17 ms	2827 ms	762 ms	1740 ms
Patch Weights	464 ms	2 ms	1 ms	1 ms	60 ms	69 ms	26 ms
Macro Opt.	-	-	-	1174 ms	353 ms	15 622 ms	18 190 ms
Meso Opt.	-	-	-	-	23 112 ms	4485 ms	3510 ms
<b>Rendering</b>							
Depth Pre-Pass	0.325 ms	0.023 ms	0.017 ms	0.026 ms	0.059 ms	0.092 ms	0.048 ms
Max-Blending	0.410 ms	0.095 ms	0.085 ms	0.119 ms	0.120 ms	0.166 ms	0.112 ms
Accumulate	0.817 ms	0.217 ms	0.295 ms	0.443 ms	0.200 ms	0.277 ms	0.187 ms
Resolve	0.355 ms	0.099 ms	0.162 ms	0.332 ms	0.081 ms	0.096 ms	0.152 ms
Sum	1.906 ms	0.434 ms	0.559 ms	0.919 ms	0.460 ms	0.631 ms	0.499 ms
Atlas Rendering	0.183 ms	0.054 ms	0.059 ms	0.133 ms	0.063 ms	0.087 ms	0.065 ms

**4.2.2 Rendering Time.** As the *depth pre-pass* renders the input geometry into the depth buffer, its cost varies only with the number of rendered triangles (*render faces* in Table 1). Note that the number of faces is lower if smaller overlaps are computed. The *max-blending* and *accumulate* stages, which perform per-fragment blending, are the most expensive parts. Finally, the *resolve* stage is a post-process that is invoked at most once per screen space pixel. Performance thus depends on how much of the screen is covered. Besides dividing the accumulated values (e.g. albedo and normals) by the respective weight and de-gaussianizing the results, lighting computations are performed.

For further optimization, one can limit the blending candidates by assigning a fixed number of texture coordinates and weights per vertex (e.g. 3). For this, the most influential patches for every vertex have to be determined prior to rendering. By doing so, the above render stages can be transformed into an inexpensive single-pass algorithm. The drop shadows in the results are merely for artistic purposes and are not considered for the rendering time.

The last row of Table 1 contains the rendering cost for all objects using texture atlases instead of the direct rendering method. Note that, apart from the geometry-heavy ice armadillo, the cost is bounded by the fragment operations. The depicted objects cover the screen space by approximately 10 – 35%. We refer to the rendering results in the supplemental material for examples. Moving the camera toward the rendered objects until all screen fragments are covered, increases overall atlas rendering timings to 0.2 ms – 0.6 ms. Due to mipmapping, the actual size of the atlas does not impact the rendering performance significantly.



Table 2. Timings for a parallelized atlas generation for the bunny mesh (6 CPU cores, 149 patches). This includes atlases for albedo, normal, roughness and height texture.

Input sizes	Output sizes			
	1024 <sup>2</sup>	2048 <sup>2</sup>	4096 <sup>2</sup>	8192 <sup>2</sup>
1024 <sup>2</sup>	0.37 s	0.71 s	2.03 s	6.90 s
8192 <sup>2</sup>	0.39 s	0.82 s	2.51 s	7.29 s

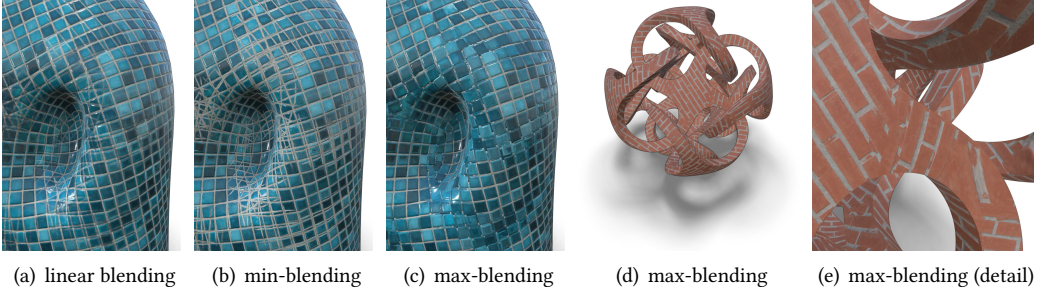


Fig. 16. Failure cases of our method: (a-c) meso-alignment fails to adequately match regular structures for large Gaussian curvature. None of the histogram-preserving blending strategies conceals the misalignment. (d-e) Success of the meso optimization depends on the number of matching interest points and mesh topology. Even if some alignment errors can be remedied, rotation and scaling artifacts are noticeable. Ghost model from [Thingiverse 2011a], entangled model from [Thingiverse 2015], and textures from [Textures.com 2020].

**4.2.3 Atlas Generation Time.** Table 2 shows exemplary timings for the texture atlas creation of a PBR set consisting of albedo, normal, roughness and height textures. The corresponding mesh was partitioned into 149 patches.

**4.2.4 Memory Requirements for Rendering.** Assuming 1 byte per channel, atlases for the aforementioned textures consume 8 bytes per texel, which is e.g. 0.5 GB for an  $8192 \times 8192$  atlas set. Alternatively, for the direct rendering approach, every vertex needs to store one texture coordinate and one weight value for each patch that covers it. The unprocessed armadillo mesh has 172 974 vertices and the render mesh from Figure 11 consists of 439 654 vertices. Assuming 12 single precision floats for position, normal, tangent, texture coordinate, and patch weight per vertex, we need to store roughly 14 MB additional data. As many vertex positions, tangents, and normals are shared among different patches, a possible optimization would be to store them in a Shader Storage Buffer Object to reduce redundancy.

### 4.3 Limitations and Future Work

Our method works on a wide variety of surfaces but degenerates in certain situations. Meshes with extreme Gaussian curvature or non-developable fine geometric detail result in high, visible distortion after parametrization. While decreasing the patch size in affected regions does help, patches that are too small also jeopardize the synthesis quality.

Material textures from fully stochastic to almost regular are handled well by our method though certain types of textures do not achieve the target quality. Fully regular textures cannot be mapped properly if the mesh contains non-negligible Gaussian curvature without causing visible distortion. If they are too regular, the micro step has no leeway to fix patch transitions. Similarly, some textures

have additional semantic constraints that are inadequately captured by our alignment and matching procedures, for example cellular textures with distinct cell types that are not allowed to mix. Some cases are shown in Figure 16. We plan to improve the handling of these challenging textures in the future by adding additional constraints to the meso optimization, ideally automatically derived from the texture content.

Currently, the only opportunity for user interaction is after the meso step. The micro step can be executed during rendering and is fast enough that users can transform individual patches in real-time. In the future we plan to explore UI metaphors that enable an intuitive and iterative workflow where designers are cooperating with our complete pipeline.

Another avenue for research is to expand the scope of our method to handle multiple materials as well as incrementally re-texturing mesh parts that changed, e.g. due to CSG operations or sculpting.

## 5 CONCLUSION

We devised a multi-stage texture synthesis method that is able to create high quality texture maps for 3D surfaces. The method starts by growing patches with limited geodesic radius and performs Lloyd relaxation steps to approximate a centroidal Voronoi tessellation. By employing parametrization methods to compute patch texture coordinates, complex geometric shapes can be mapped with low distortion.

We support a broad spectrum of textures, from stochastic to near-regular, by leveraging content-awareness on multiple scales. Our macro step optimizes global directional alignment to account for strong directional patterns present in many textures. The meso step aligns interest points in the textures and performs a slight cubic warping of patches to improve content alignment between neighboring patches. While this re-introduces some distortion, the improved synthesis quality is worth the trade-off.

The macro and meso steps improve the starting conditions for the micro step. Due to the optimized content alignment, the histogram-preserving blending strategy produces convincing results in most cases, so that expensive pixel-based synthesis or seam optimization schemes are not required. In addition to *linear* histogram-preserving blending, we argue that for PBR texture sets a *min*- or *max*-blending scheme often results in a more natural transition between structures.

The result of our method can be baked into a texture atlas, which is especially simple because the patches are already parametrized and have roughly uniform size. However, this tends to require large textures with partially redundant content for bigger meshes, thus we propose an inexpensive multi-pass method to perform the blending in screen-space directly during rendering.

With the presented method, a wide class of 2D texture samples can be mapped onto 3D objects in an automatic and seamless fashion, with no visible repetitions and low overall distortion.

## ACKNOWLEDGMENTS

This work was partially funded by the German Federal Ministry of Education and Research within the “eVerest” project under the funding code 02P14A146 and the “VirtualDisaster” project under the funding code 13N15155. Furthermore, this work was partially funded by the European Regional Development Fund within the “HDV-Mess” project under the funding code EFRE-0500038.

## REFERENCES

- Paul J. Besl and Neil D. McKay. 1992. A Method for Registration of 3-D Shapes. *IEEE Trans. Pattern Anal. Mach. Intell.* 14, 2 (Feb. 1992), 239–256.
- David Bommes, Marcel Campen, Hans-Christian Ebke, Pierre Alliez, and Leif Kobbelt. 2013. Integer-grid maps for reliable quad meshing. *ACM Transactions on Graphics (TOG)* 32, 4 (2013), 98.
- David Bommes, Henrik Zimmer, and Leif Kobbelt. 2009. Mixed-integer Quadrangulation. *ACM Trans. Graph.* 28, 3, Article 77 (July 2009), 10 pages.
- David Bommes, Henrik Zimmer, and Leif Kobbelt. 2010. Practical mixed-integer optimization for geometry processing. In *International Conference on Curves and Surfaces*. Springer, 193–206.
- Brent Burley. 2019. On Histogram-Preserving Blending for Randomized Texture Tiling. *Journal of Computer Graphics Techniques (JCGT)* 8, 4 (8 November 2019), 31–53.
- Marcel Campen, David Bommes, and Leif Kobbelt. 2015. Quantized global parametrization. *ACM Transactions on Graphics (TOG)* 34, 6 (2015), 192.
- Thomas Deliot and Eric Heitz. 2019. Procedural Stochastic Textures by Tiling and Blending. In *GPU Zen 2: Advanced Rendering Techniques*, W. Engel (Ed.). Black Cat Publishing Inc., 144 West D Street Suite 204, Encinitas, CA 92009, Chapter 2 in Part IV, 177–200.
- Yue Dong, Sylvain Lefebvre, Xin Tong, and George Drettakis. 2008. Lazy Solid Texture Synthesis. In *Proceedings of the Nineteenth Eurographics Conference on Rendering (EGSR '08)*. Eurographics Association, Aire-la-Ville, Switzerland, Switzerland, 1165–1174.
- Qiang, Du, Vance. Faber, and Max. Gunzburger. 1999. Centroidal Voronoi Tessellations: Applications and Algorithms. *SIAM Rev.* 41, 4 (1999), 637–676.
- Jérémie Dumas, An Lu, Sylvain Lefebvre, Jun Wu, and Christian Dick. 2015. By-example Synthesis of Structurally Sound Patterns. *ACM Trans. Graph.* 34, 4, Article 137 (July 2015), 12 pages.
- Alexei A. Efros and William T. Freeman. 2001. Image Quilting for Texture Synthesis and Transfer. In *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '01)*. ACM, New York, NY, USA, 341–346.
- Alexei A. Efros and Thomas K. Leung. 1999. Texture Synthesis by Non-Parametric Sampling. In *Proceedings of the International Conference on Computer Vision-Volume 2 - Volume 2 (ICCV '99)*. IEEE Computer Society, Washington, DC, USA, 1033–.
- Martin A. Fischler and Robert C. Bolles. 1981. Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography. *Commun. ACM* 24, 6 (June 1981), 381–395.
- Bruno Galerne, Ares Lagae, Sylvain Lefebvre, and George Drettakis. 2012. Gabor noise by example. *ACM Transactions on Graphics (TOG)* 31, 4 (2012), 73.
- Leon Gatys, Alexander S Ecker, and Matthias Bethge. 2015. Texture synthesis using convolutional neural networks. In *Advances in neural information processing systems*. 262–270.
- Guillaume Gilet, Basile Sauvage, Kenneth Vanhoey, Jean-Michel Dischler, and Djamchid Ghazanfarpour. 2014. Local random-phase noise for procedural texturing. *ACM Transactions on Graphics (TOG)* 33, 6 (2014), 195.
- Gabriele Gorla, Victoria Interrante, and Guillermo Sapiro. 2001. Growing fitted textures. *SIGGRAPH 2001 Sketches and Applications* (2001), 191.
- Gael Guennebaud, Benoît Jacob, et al. 2010. Eigen v3. <http://eigen.tuxfamily.org>.
- Jianwei Han, Kun Zhou, Li-Yi Wei, Minmin Gong, Hujun Bao, Xinming Zhang, and Baining Guo. 2006. Fast example-based surface texture synthesis via discrete optimization. *The Visual Computer* 22, 9-11 (2006), 918–925.
- Christopher G Harris, Mike Stephens, et al. 1988. A combined corner and edge detector.. In *Alvey vision conference*, Vol. 15. Citeseer, 10–5244.
- David J. Heeger and James R. Bergen. 1995. Pyramid-based Texture Analysis/Synthesis. In *Proceedings of the 22Nd Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '95)*. ACM, New York, NY, USA, 229–238.
- Eric Heitz and Fabrice Neyret. 2018. High-Performance By-Example Noise Using a Histogram-Preserving Blending Operator. *Proc. ACM Comput. Graph. Interact. Tech.* 1, 2, Article 31 (Aug. 2018), 25 pages.
- Alec Jacobson, Daniele Panozzo, et al. 2018. libigl: A simple C++ geometry processing library. <https://libigl.github.io/>.
- Nikolay Jetchev, Urs Bergmann, and Roland Vollgraf. 2016. Texture synthesis with spatial generative adversarial networks. *arXiv preprint arXiv:1611.08207* (2016).
- Jiaya Jia, Jian Sun, Chi-Keung Tang, and Heung-Yeung Shum. 2006. Drag-and-drop Pasting. *ACM Trans. Graph.* 25, 3 (July 2006), 631–637.
- Johannes Kopf, Chi-Wing Fu, Daniel Cohen-Or, Oliver Deussen, Dani Lischinski, and Tien-Tsin Wong. 2007. Solid Texture Synthesis from 2D Exemplars. *ACM Trans. Graph.* 26, 3, Article 2 (July 2007).
- Vivek Kwatra, Arno Schödl, Irfan Essa, Greg Turk, and Aaron Bobick. 2003. Graphcut textures: image and video synthesis using graph cuts. In *ACM Transactions on Graphics (ToG)*, Vol. 22. ACM, 277–286.



- Ares Lagae, Peter Vangorp, Toon Lenaerts, and Philip Dutré. 2010. Procedural Isotropic Stochastic Textures by Example. *Computers & Graphics (Special issue on Procedural Methods in Computer Graphics)* 34, 4 (2010), 312–321.
- Anass Lasram and Sylvain Lefebvre. 2012. Parallel Patch-based Texture Synthesis. In *Proceedings of the Fourth ACM SIGGRAPH / Eurographics Conference on High-Performance Graphics (EGGH-HPG'12)*. Eurographics Association, Goslar Germany, Germany, 115–124.
- Sylvain Lefebvre and Hugues Hoppe. 2005. Parallel Controllable Texture Synthesis. In *ACM SIGGRAPH 2005 Papers (SIGGRAPH '05)*. ACM, New York, NY, USA, 777–786.
- Sylvain Lefebvre and Hugues Hoppe. 2006. Appearance-space Texture Synthesis. In *ACM SIGGRAPH 2006 Papers (SIGGRAPH '06)*. ACM, New York, NY, USA, 541–548.
- Bruno Lévy, Sylvain Petitjean, Nicolas Ray, and Jérôme Maillot. 2002. Least Squares Conformal Maps for Automatic Texture Atlas Generation. *ACM Trans. Graph.* 21, 3 (July 2002), 362–371.
- Andrew Nealen and Marc Alexa. 2003. Hybrid Texture Synthesis. In *Proceedings of the 14th Eurographics Workshop on Rendering (EGRW '03)*. Eurographics Association, Aire-la-Ville, Switzerland, Switzerland, 97–105.
- Marcin Novotni, Reinhard Klein, et al. 2002. Computing geodesic distances on triangular meshes. In *In Proc. of WSCG'2002*.
- Patrick Pérez, Michel Gangnet, and Andrew Blake. 2003. Poisson Image Editing. *ACM Trans. Graph.* 22, 3 (July 2003), 313–318.
- Ken Perlin. 1985. An image synthesizer. *ACM Siggraph Computer Graphics* 19, 3 (1985), 287–296.
- Nico Pietroni, Paolo Cignoni, Miguel Otaduy, and Roberto Scopigno. 2010. Solid-texture synthesis: a survey. *IEEE Computer Graphics and Applications* 30, 4 (2010), 74–89.
- Javier Portilla and Eero P. Simoncelli. 2000. A Parametric Texture Model Based on Joint Statistics of Complex Wavelet Coefficients. *International Journal of Computer Vision* 40, 1 (01 Oct 2000), 49–70.
- Emil Praun, Adam Finkelstein, and Hugues Hoppe. 2000. Lapped Textures. In *Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '00)*. ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, 465–470.
- Michael Rabinovich, Roi Poranne, Daniele Panozzo, and Olga Sorkine-Hornung. 2017. Scalable Locally Injective Mappings. *ACM Trans. Graph.* 36, 4, Article 37a (April 2017).
- Ryan Schmidt, Cindy Grimm, and Brian Wyvill. 2006. Interactive Decal Compositing with Discrete Exponential Maps. *ACM Trans. Graph.* 25, 3 (July 2006), 605–613.
- Stanford. 2020. The Stanford 3D Scanning Repository. <http://www-graphics.stanford.edu/data/3Dscanrep/>. Accessed: 2020-03-13.
- Textures.com. 2020. Textures for 3D, graphic design and Photoshop. <https://www.textures.com/>. Accessed: 2019-12-12.
- Thingiverse. 2011a. Ghost Model by navalguijo on Thingiverse. <https://www.thingiverse.com/thing:12585> Accessed March 17, 2020.
- Thingiverse. 2011b. Part of Sesame Street Sign Model by cptnAWESOME on Thingiverse. <https://www.thingiverse.com/thing:14974> Accessed March 17, 2020.
- Thingiverse. 2012. Anvil Model by WorksBySolo on Thingiverse. <https://www.thingiverse.com/thing:31397> Accessed March 17, 2020.
- Thingiverse. 2015. Metatron/Entangled Model by bathsheba on Thingiverse. <https://www.thingiverse.com/thing:1146870> Accessed March 17, 2020.
- Xin Tong, Jingdan Zhang, Ligang Liu, Xi Wang, Baining Guo, and Heung-Yeung Shum. 2002. Synthesis of Bidirectional Texture Functions on Arbitrary Surfaces. *ACM Trans. Graph.* 21, 3 (July 2002), 665–672.
- Greg Turk. 2001. Texture Synthesis on Surfaces. In *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '01)*. ACM, New York, NY, USA, 347–354.
- Dmitry Ulyanov, Vadim Lebedev, Andrea Vedaldi, and Victor S Lempitsky. 2016. Texture Networks: Feed-forward Synthesis of Textures and Stylized Images.. In *ICML*, Vol. 1. 4.
- Li-Yi Wei, Sylvain Lefebvre, Vivek Kwatra, and Greg Turk. 2009. State of the art in example-based texture synthesis.
- Li-Yi Wei and Marc Levoy. 2000. Fast Texture Synthesis Using Tree-structured Vector Quantization. In *Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '00)*. ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, 479–488.
- Li-Yi Wei and Marc Levoy. 2001. Texture Synthesis over Arbitrary Manifold Surfaces. In *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '01)*. ACM, New York, NY, USA, 355–360.
- Jingdan Zhang, Kun Zhou, Luiz Velho, Baining Guo, Heung-Yeung Shum, Heung-Yeung Shum, and Heung-Yeung Shum. 2003. Synthesis of Progressively-variant Textures on Arbitrary Surfaces. *ACM Trans. Graph.* 22, 3 (July 2003), 295–302.